A Self-adaptive Coevolutionary Algorithm

A Self-adaptive Coevolutionary Algorithm

Mario Hevia Fajardo*
m.heviafajardo@bham.ac.uk
University of Birmingham, UK

Jamal Toutouh
jamal@uma.es
ITIS, Software, University of Malaga, Spain

Erik Hemberg*
Una-May O'Reilly
hembergerik@csail.mit.edu
unamay@csail.mit.edu
MIT, USA

Per Kristian Lehre
p.k.lehre@bham.ac.uk
University of Birmingham, UK

ABSTRACT

Coevolutionary algorithms are helpful computational abstractions of adversarial behavior and they demonstrate multiple ways that populations of competing adversaries influence one another. We introduce the ability for each competitor's mutation rate to evolve through self-adaptation. Because dynamic environments are frequently addressed with self-adaptation, we set up dynamic problem environments to investigate the impact of this ability. For a simple bilinear problem, a sensitivity analysis of the adaptive method's parameters reveals that it is robust over a range of multiplicative rate factors, when the rate is changed up or down with equal probability. An empirical study determines that each population's mutation rates converge to values close to the error threshold. Mutation rate dynamics are complex when both populations adapt their rates. Large scale empirical self-adaptation results reveal that both reasonable solutions and rates can be found. This addresses the challenge of selecting ideal static mutation rates in coevolutionary algorithms. The algorithm's payoffs are also robust. They are rarely poor and frequently they are as high as the payoff of the static rate to which they converge. On rare runs, they are higher.

CCS CONCEPTS

• **Theory of computation** → **Evolutionary algorithms**; **Adversary models**; • **Security and privacy** → *Vulnerability management*.

KEYWORDS

coevolution, cyber security, evolutionary algorithms

ACM Reference Format:
Mario Hevia Fajardo, Jamal Toutouh, Erik Hemberg, Una-May O'Reilly, and Per Kristian Lehre. 2024. A Self-adaptive Coevolutionary Algorithm. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3638529.3654132

*Corresponding authors.

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0494-9/24/07
https://doi.org/10.1145/3638529.3654132

1 INTRODUCTION

Coevolutionary algorithms are useful for abstractly modeling real world contexts such as cyber security. In these contexts, one sees large scale numbers of attackers and defenders engaged in an arms race or driving one side to extinction. This phenomena aligns well with the coevolutionary algorithm's competing populations, selection, variation, and coevolutionary dynamics. Given how intelligently cyber adversaries can adapt, it is arguable they do not simply adapt, but they also change their effective rate of adapation. In evolutionary algorithms (EAs), self-adaptation is a parameter control mechanism where mutation rates, or other parameters, are encoded and evolved within genomes. It has been shown to be effective in classical EA, i.e., those with only one population [22]. The argument for self-adaption is stronger for dynamic environments, intuitively because their non-stationarity requires more rate flexibility. In one sense, in a competitive coevolutionary algorithm (CCA) each population's adversaries are similarly non-stationary, so they mimic a dynamic environment. This raises the question of how one or both populations using self-adapation changes overall behavior (e.g., rates and payoffs)? The question of the consequences of dynamic variations of the competition environment, e.g., optima or budget changes, within the context of self-adaptation, also arises.

In this contribution we specifically investigate the impact of self-adapting mutation rates for CCAs. Self-adaptation offers the advantage that a schedule of changes devised prior to a run is unnecessary. The genome encoding behavior is extended with a gene that encodes the rate. The self-adapting gene/rate is passed from parent to offspring through inheritance, hence it is subject to selective pressure. It undergoes mutation each generation after replication, and is then used when mutating the rest of the genome. This effectively tunes the gene to an ideal rate relative to the fitness function. Consider that in a classical EA, in different areas of the search space, some mutation rates can be more helpful than others in optimizing payoffs or time to convergence. Additionally, non-elitist EAs need to use mutation rates below the *error threshold* [17, 24]. An error threshold is an attribute of a non-elitist EA. Informally, the threshold describes how the performance of the algorithm degrades when the level of mutation is increased beyond a point which overwhelms the selective pressure. In a problem where the optima changes, there is added value to being able to change a rate. The central challenge of self-adaptive EAs however, has been to find a robust adaptation mechanism [6]. Next, consider that in a CCA, given its competitive (minimax) paradigm, the competing

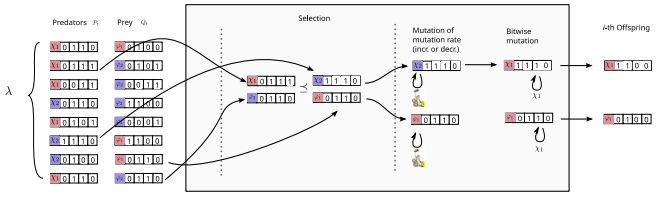


Figure 1: Overview of a self-adaptive CCA called the SA-PDCoEA. There are two populations (called Predator and Prey). A solution contains both a strategy and a mutation rate. The fitness depends only on the strategy. All offspring are produced identically and independently. The mutation rate of a selected solution is first randomly changed and then the strategy with the new mutation rate. This creates a new solution with changed mutation rate and strategy.

population constantly changes. Regardless of a static or dynamic environment, there is thus value to being able to change a rate. When the environment also changes, the value of changing the rate of mutation increases even more [19]. Meanwhile, mutation rates are mostly static [16, 18, 19] in CCA designs and it remains unclear how to chose the correct mutation rate [18, 19]. Arguably, a self-adapting gene/mutation rate in a CCA could advantageously hone into both the other population's behavior and the environment and resolve the difficult requirement for a schedule.

In this study we transfer a theoretically-sound method of self-adaptating mutation rates from EA research to a competitive coevolutionary algorithm called SA-PDCoEA [6, 18], see Figure 1. Predicting or theoretically analyzing what will happen with SA-PDCoEA is extremely challenging. On top of the intrinsic complexity of CCAs, a self-adapting adversary poses further challenges because how much the solutions change is in flux. We formulate the following questions and undertake an empirical analysis to answer them: **RQ-1** Can the mutation rates self-adapt to within the empirical error threshold? This has implications for the SA-PDCoEA runtime. **RQ-2** What are the dynamics of the mutation rates when one or both populations self-adapt their mutation rates? **RQ-3** What are the payoff impacts of self-adaptive mutation rates?

Our contributions are the following: • We introduce a self-adaptation method for mutation rates in CCAs, by way of SA-PDCoEA. This method is demonstrated to support the mutation rate adapting to suit the environment of each problem we study.

• In combination, we analyze two parameters of the adaptive mutation rate method: probability of incrementing (or decrementing) the rate, p_{inc}, and the multiplicative factor by which the rate is changed, A. We find that their impact differs depending the problem environment. With a BILINEAR problem environment, within the $p_{inc} \times A$ parameter space, results are best when an incremental change to the mutation rate is equally likely as a decrement, i.e., $p_{inc} = 0.5$. For $p_{inc} = 0.5$ many change factors are equally effective, making their selection less critical.

• We analyze the SA-PDCoEA on the BILINEAR problem and observe that each population's mutation rates converge to values close but below the error threshold. This suggests that the mutation rate can adapt to the problem without previous knowledge.

• Mutation rate dynamics are very complex when both populations adapt their rates. In both the problem environments we study, we observe that mutation rates of the two populations co-adapt with each other. Empirical results on DefendIt-B reveal that, with self-adaptation, both reasonable solutions (in this case high payoffs) and rates can be found by the algorithm. This showcases that self-adaptation can address the very challenging problem of selecting appropriate static mutation rates in coevolutionary algorithms.

• We observe that mutation rates coevolve to mutation rates giving highest payoff. The attacker mutation rate depends on the adversary's mutation rate. On DefendIt-B SA-PDCoEA can obtain similar or better payoffs as PDCoEA when PDCoEA's mutation rate has been chosen well. Thus the algorithm's payoffs are robust. They are rarely poor and often as high, and sometimes higher, as the payoffs of runs using the static rate to which they converge.

The paper is structured as follows. Section 2 presents related work. Section 3 describes SA-PDCoEA and the problem environments we investigate. Section 4 presents the empirical experiments and results. Finally, Section 5 draws conclusions and future work.

2 RELATED WORK

We briefly present work related to competitive coevolution and self-adaptation. Biological coevolution refers to the influences two or more species exert on each other's evolution [9, 27]. Coevolution can be mutually beneficial (cooperative) or adversarial (competitive). The competition can arise from e.g., constrained and shared resources or predator-prey relationships.

An EA typically evolves individual solutions, e.g., fixed length bit strings as in Genetic Algorithms (GAs) [12] with an *a-priori* defined fitness function to evaluate an individual's quality. In contrast, in coevolutionary algorithms an individual's fitness is based on interactions with other individuals or a dynamic environment to mimic coupled biological species-to-species interactions.

We extend a growing body of work on coevolutionary algorithms [2, 13, 16, 23, 25–27, 29]. Our focus is on Competitive Coevolutionary Algorithms (CCAs). In a basic CCA at each generation an individual's fitness score is based on its performance outcomes in its competitions. E.g., the sum of the performance outcomes or the average, maximum, minimum, or median outcome [3].

Variations of the CCA have been defined for many specific problem domains, e.g., [1, 5, 11, 20, 21, 30]. Different games or simplified problems have been studied [3, 10, 15, 16]. The extension to CCAs of FlipIt to DefendIt was motivated by application of CCAs that model security scenarios such as in cyber-networks [14, 19].

There is some theoretical and empirical analysis of coevolutionary algorithms regarding *error thresholds* [17, 18, 24]. The runtime of non-elitist EAs can drastically increase from polynomial to exponential when the mutation rate is increased above the error threshold [17]. Error thresholds have been studied theoretically in CCAs [18], and also recently been estimated empirically [14].

Self-adaptation has a long history in evolutionary computation (see [22] for a survey) [4, 28]. In a first runtime analysis demonstrating the benefit of self-adaptation in population-based EAs, it was shown that an EA with (μ, λ)-selection and self-adaptation can escape a local optimum, while the same algorithm with a fixed mutation rate below the error threshold does not escape the local optimum in polynomial time, or with a fixed mutation rate above the error threshold cannot reach the global optimum in polynomial time [7]. This analysis showed that when the algorithm chooses between two mutation rates, it selects a high mutation rate to escape the local optimum, and a low mutation rate when close to the optimum. In subsequent work it was shown that a non-elitist EA with

(μ, λ)-selection and the same self-adaptation mechanism as used in this paper can adapt the mutation rate to a problem with unknown structure. Other work has investigated a self-adaptive $(1, \lambda)$ EA [8], showing asymptotically better runtime than the classic $(1, \lambda)$ EA. The $(1, \lambda)$ EA has a parent population size of one, and the adaptive mutation rate therefore becomes a global parameter inherited by all offspring. We investigate the impact of self-adaptation in CCAs with large parent and offspring population sizes.

3 PRELIMINARIES

3.1 Self-Adaptive Pairwise Dominance CoEA

We now describe a self-adaptive CCA called SA-PDCoEA, see also Algorithm 1 and Figure 1. Informally, each individual has its own mutation rate inherited from its parent, i.e., evolving with the search points, which is used to mutate its genome. SA-PDCoEA follows the mechanism described by [6].

Algorithm 1 Self-adaptive Pairwise Dominance CoEA [18]

Require: Population size $\lambda \in \mathbb{N}$.
Require: Payoff function $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ with $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$.
Require: Parameters $p_{\text{inc}} \in (0,1)$, $A \in (1, \infty)$, and $\chi_{\min} \in (0, n)$.
Require: Initial populations $(P_0, Q_0) \in (\mathcal{X} \times (0,n))^\lambda \times (\mathcal{Y} \times (0,n))^\lambda$

for $t \in \mathbb{N}$ until termination criterion met **do**
  for $i \in [\lambda]$ **do**
    Sample $(x_1, \chi_1), (x_2, \chi_2) \sim \text{Unif}(P_t)$
    Sample $(y_1, \varphi_1), (y_2, \varphi_2) \sim \text{Unif}(Q_t)$
    if $(x_1, y_1) \succeq_g (x_2, y_2)$ **then**
      $(x, \chi) := (x_1, \chi_1)$ and $(y, \varphi) := (y_1, \varphi_1)$
    else $(x, \chi) := (x_2, \chi_2)$ and $(y, \varphi) := (y_2, \varphi_2)$
    $$\chi' := \begin{cases} \min(A\chi, n) & \text{with probability } p_{\text{inc}} \\ \max(\chi/A, \chi_{\min}) & \text{otherwise.} \end{cases}$$
    $$\varphi' := \begin{cases} \min(A\varphi, n) & \text{with probability } p_{\text{inc}} \\ \max(\varphi/A, \chi_{\min}) & \text{otherwise.} \end{cases}$$
    Obtain x' by flipping each bit in x with prob. χ'/n.
    Obtain y' by flipping each bit in y with prob. φ'/n.
    Set $P_{t+1}(i) := (x', \chi')$ and $Q_{t+1}(i) := (y', \varphi')$.

More precisely, the algorithm assumes maximin-optimization of a payoff-function $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, i.e., finding a "predator" x which maximizes the function $f(x) := \min_{y \in \mathcal{Y}} g(x, y)$, i.e., its payoff against its worst-case prey. Each individual (x, χ) in the predator population P consists of a search point (or genotype) $x \in \mathcal{X}$ and a mutation rate $\chi \in (0, n)$. Similarly, each individual (y, φ) in the prey population Q consists of a search point $y \in \mathcal{Y}$ (or genotype) and a mutation rate $\varphi \in (0, n)$. Note that we use the term *mutation probability* in bitwise mutation to denote the probability of flipping a single bit. Given a bitstring of length n and a mutation probability χ/n, then we call the value χ the *mutation rate*. In order for χ/n to be a probability, the mutation rate must satisfy $\chi \in [0, n]$.

In each generation, see Figure 1, the algorithm produces λ pairs of predators and prey identically and independently as follows. Two predators are first selected by sampling uniformly at random , (x_1, χ_1) and (x_2, χ_2). Two prey (y_1, φ_1) and (y_2, φ_2) are similarly selected. Similarly to PDCoEA, (x_1, y_1) is kept if it dominates (x_2, y_2), otherwise (x_2, y_2). The domination relation $\succeq_g$ is not the Pareto dominance, instead we say that $(x_1, y_1) \succeq_g (x_2, y_2)$ if and only if $g(x_1, y_2) \geq g(x_1, y_1) \geq g(x_2, y_1)$. Informally, this means that the predator x_1 is better than predator x_2 when evaluated against prey y_1, and at the same time, prey y_1 is better than prey y_2 when evaluated against predator x_1. See [18] for further discussion on the dominance relation. The selected pair is then mutated in two steps. First, the algorithm "mutates" the mutation rates χ and φ of the selected individuals, by either increasing the mutation rate by a factor $A > 1$ (with probability p_{inc}), or decreasing the mutation rate by multiplying by a factor $1/A < 1$ (with probability $1 - p_{\text{inc}}$). Finally, the new search points x' and y' are obtained by mutating x and y with their new respective mutation rates χ' and φ'. Compared to the PDCoEA, there are three additional parameters: A which is the mutation rate increment factor, p_{inc} which is the probability to increase the mutation rate, and $\chi_{\min}$ which is the minimal mutation rate allowed of any individual.

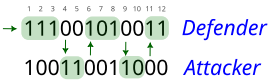


Figure 2: DefendIt-B on a single resource. Initially, the resource is held by the defender. At time step 1, both the defender and the attacker attempt to acquire the resource, and ownership remains with the defender. At time step 4, the attacker acquires the resource. The defender re-acquires the resource at time step 6 and keeps it until time step 9 when the attacker regains ownership. The resource changes hand a final time at time step 11. At the end of the game, the defender has owned the resource for 8 time steps, while the attacker has owned it for 4 time steps. Note, in DefendIt-B, in contrast to DefendIt, if the defender was over budget the attacker could acquire the resource at step 8.

3.2 Problem environments

3.2.1 Bilinear. This is a simple class of maximin-optimization problems with a clear structure [18] defined for $\alpha, \beta \in (0,1)$ by

$$\mathit{Bilinear}(x, y) := |y|(|x| - \beta n) - \alpha n|x|, \tag{1}$$

where for any bitstring $z \in \{0,1\}^n$, $|z| := \sum_{i=1}^n z_i$ denotes the number of 1-bits in z. Assuming that the prey always responds with an optimal decision for every $x \in \mathcal{X}$, the predator gets the unimodal function f which has maximum when $|x| = \beta n$

$$f(x) := \min_{y \in \{0,1\}^n} g(x, y) = \begin{cases} |x|(1 - \alpha n) - \beta n & \text{if } |x| \leq \beta n \\ -\alpha n|x| & \text{if } |x| \geq \beta n \end{cases} \tag{2}$$

3.2.2 DefendIt-B. We modify the NP-hard DefendIt game from [19] to prevent actions when the budget is overspent and call it DefendIt-B. Note that the difference lies in the calculation of resource ownership in scenarios where the opponent's budget surpasses its limit. Figure 2 shows an example of DefendIt-B.

An instance of the DefendIt-B game is given by a tuple $(k, \ell, v, c, B^D, B^A)$ where $k \in \mathbb{N}$ is the number of resources, $\ell \in \mathbb{N}$ is the number of time-steps, $v = (v^{(1)}, \ldots, v^{(k)})$ where $v^{(j)} \in [0, \infty)$ is the value of resource $j \in [k]$, $c = (c^{(1)}, \ldots, c^{(k)})$ where $c^{(j)} \in [0, \infty)$ is the cost of resource $j \in [k]$, $B^D \in [0, \infty)$ is the defender's budget, and $B^A \in [0, \infty)$ is the attacker's budget.

Defender and attacker strategies are represented by bitstrings of length $n := k \cdot \ell$. We adopt the notation $x = (x_1^{(1)}, \ldots, x_\ell^{(1)}, \ldots, x_1^{(k)}, \ldots, x_\ell^{(k)}) \in \{0,1\}^n$ for the defender's strategy, where $x_i^{(j)} = 1$ for $j \in [k]$ and $i \in [\ell]$ means that the defender attempts to acquire resource j at time i. Analogously,

we denote $y = (y_1^{(1)}, \ldots, y_\ell^{(1)}, \ldots, y_1^{(k)}, \ldots, y_\ell^{(k)}) \in \{0,1\}^n$ for the attacker's strategy, where $y_i^{(j)} = 1$ for $j \in [k]$ and $i \in [\ell]$ means that the attacker attempts to acquire resource j at time i.

We define the payoff of strategies in terms of the resources ownership. In particular, $z_i^{(j)} \in \{0,1\}$ is the ownership of resource $j \in [k]$ at time $i \in \{0\} \cup [\ell]$, where $z_i^{(j)} = 1$ indicates that the defender owns resource j at time i, and $z_i^{(j)} = 0$ means that the attacker owns resource j at time i. For all $j \in [k]$, we define $z_0^{(j)}(x,y) := 1$, which corresponds to the assumption that the defender is in possession of all resources at the beginning of the game.

In DefendIt-B when a player attempts to acquire the resource while the opponent does not, the player obtains the resource. If neither the defender or attacker move, the ownership does not change. If both the defender and the attacker attempt to acquire the resource, the ownership does not change. The ownership of a resource j is defined inductively for $i \in [\ell]$ as follows

$$z_i^{(j)}(x,y) := \begin{cases} z_{i-1}^{(j)}(x,y) & \text{if } x_i^{(j)} = y_i^{(j)} \text{ and } C(x) \le B^D, \\ z_{i-1}^{(j)}(x,y) & \text{if } x_i^{(j)} = y_i^{(j)} = 0, \\ 1 & \text{if } x_i^{(j)} = 1 \text{ and } y_i^{(j)} = 0 \text{ and } C(x) \le B^D, \\ 0 & \text{if } x_i^{(j)} = 0 \text{ and } y_i^{(j)} = 1 \text{ and } C(y) \le B^A. \end{cases}$$

The overall cost of a defender or an attacker strategy x is $C(x) := \sum_{j=1}^{k} c^{(j)} \sum_{i=1}^{\ell} x_i^{(j)}$,i.e., the number attempts of acquiring a resource weighted by the cost of that resource. A defender strategy x is called *over-budget* if $C(x) > B^D$. Similarly, an attacker strategy y is called *over-budget* if $C(y) > B^A$. Finally, the payoff function g_1 (g_2) for the defender (attacker) are defined as

$$g_1(x,y) := \begin{cases} \sum_{j=1}^{k} v^{(j)} \sum_{i=1}^{\ell} z_i^{(j)}(x,y) & \text{if } C(x) \le B^D \\ -\sum_{j=1}^{k} \sum_{i=1}^{\ell} x_i^{(j)} & \text{otherwise} \end{cases}$$

$$g_2(x,y) := \begin{cases} \sum_{j=1}^{k} v^{(j)} (\ell - \sum_{i=1}^{\ell} z_i^{(j)}(x,y)) & \text{if } C(y) \le B^A \\ -\sum_{j=1}^{k} \sum_{i=1}^{\ell} y_i^{(j)} & \text{otherwise.} \end{cases}$$

Informally, if the overall cost of a strategy exceeds the player's budget (over-budget strategy), the payoff is negative and corresponds to the number of times the player attempts to acquire any resource. Note that the payoff function for an over-budget defender strategy is independent of the attacker strategy and similarly for over-budget attacker strategies. In the original DefendIt from [19] an over-budget strategy could still affect the possession of the resource. This would not affect their payoff as it would still be negative for each attempt the player makes to acquire any resource, but it could affect their opponent's payoff. In this work we prevent any actions taken after an individual exceeded their budget to affect the possession of the resource. If the overall cost of a strategy is within the budget (within-budget strategy), then the payoff is the number of time steps the player is in possession of the resource multiplied by the value of the resource. We only consider DefendIt-B instances where the cost of an item is identical to the value of the resource.

4 EXPERIMENTS

Table 1 shows the experimental design for comparing '*static*' runs without mutation rate adapation (baseline, S-S), runs where only one population adapts the mutation rate (S-A and A-S), and runs when both populations adapt the mutations rate, inducing rate co-adaptation (A-A). We run the design on two different problem environments. To directly match the classical EA's dynamic motivation for self-adaptation, we make the problem environment non-stationary. Twice after initialization, each time after some equal quantity of fitness evaluations, we change the optimum or the resource budget . This also allows us to observe the ability of the SA-PDCoEA to adapt over three different conditions.

Table 1: Self-adaptation experiment design. This is repeated for the problem environments.

Name	Predator mutation rate	Prey mutation rate
S-S	Static	Static
A-A	Adaptive	Adaptive
A-S	Adaptive	Static
S-A	Static	Adaptive

To study the self-adapting mutation rate, we set $p_{\text{inc}} = 1/2$, $A = 1.05$ and $\chi_{\min} = 0.001$ following the results from Section 4.1.1.

Bilinear. The experiments in Section 4.1.1 compare the PDCoEA with and without self-adaptation on the static Bilinear problem with problem size $n = 300$. To ensure that the problem instance has a unique maximin-optimum, we chose problem parameters $\alpha = 0$ and $\beta = 1 - 1/n$, where the optimal predator is any bitstring with exactly one zero-bit and all other bits set to one, and the optimal prey is the all-zero bitstring. To choose a population size λ, we draw inspiration from Theorem 3 in [18] which requires $\lambda = \Omega(\log n)$. We conjectured that $\lambda = 300$ would be sufficient. We chose mutation rates χ and φ in the interval $[0.01, 1.2]$ with step size 0.05. The range of the interval was chosen such that the extreme values satisfy $1.20 > \ln(2) > 0.01$. Each experiment was repeated 50 times. We recorded the number of function evaluations until the algorithm obtained the maximin-optimum or 10^7, whichever was the smallest.

The experiments in Section 4.1.1 consider whether good mutation rates can be obtained through self-adaptation. We used SA-PDCoEA with population size $\lambda = 300$. We initialized the individuals at generation $t = 0$ with different mutation rates in different runs. We used the same problem instance of Bilinear as described above.

The experiments in Section 4.1.2 investigate the dynamics of SA-PDCoEA with an emphasis on the potential benefits of using different mutation rates throughout the optimization. Therefore, we change the optima of Bilinear during the run, by changing α and β over three phases. Each run is comprised by $6 \cdot 10^7$ fitness function evaluations, and the phase (α and β) changes every $2 \cdot 10^7$ fitness function evaluations. The values of (α, β) used in each phase are, respectively: $(1/n, 1/n)$, $(1/2, 1/2)$, $(1-1/n, 1-1/n)$. For these experiments we choose $n = 400$ and $\lambda = 500$ to match the experimental setup of DefendIt-B from [19]. When self-adaptation is used, we initialize all $\chi := 0.1$ and all $\varphi := 0.1$. All strategies are initialized sampling from $\{0,1\}^n$ uniformly at random. We repeat each experiment at least 50 times.

DefendIt-B. Following Lehre et al. [19], for all experiments on DefendIt-B we use $k = 10$ number of resources and $\ell = 40$ time steps. Hence, the strategies are represented by bitstrings of length $n = k\ell = 400$. Different from [19] we set all resource costs to a uniform value of 140. The resource values and cost are identical.

In [19] it was shown that the budgets for the attacker and defender affected the error threshold for the mutation rates. Hence, in three phases, we change the budgets during the run to be: $\{B_{\text{low}} := 280, B_{\text{med}} := 2,800$, and $B_{\text{high}} := 28,000\}$ respectively.

This, combined with the cost of the resources, mean that for B_{low} only two actions are possible before exceeding the budget, 20 actions are possible for B_{med} and 200 for B_{high}.

Like Bilinear, the number of games (fitness function evaluations) in a run is $6 \cdot 10^7$ and there are three phases of $2 \cdot 10^7$ fitness evaluations each. Over the phases, the budget changes from B_{low} to B_{med} and finally to B_{high}. This range was determined by experimental observation that payoffs and mutation rates were relatively steady for a long duration in each phase. As in [19] for all experiments on DefendIt-B we choose $\lambda = 500$. Initial strategies are sampled uniformly at random from $\{0,1\}^n$. When self-adaptation is used, we initialize all $\chi := 0.1$ and all $\varphi := 0.1$ and use the fixed adaptation parameters $A = 1.05$, $p_{\text{inc}} = 0.5$, and $\chi_{\text{min}} = 0.001$ unless stated otherwise. We repeat each experiment at least 50 times.

SA-PDCoEA. When working with SA-PDCoEA, we use population size $\lambda = 300$. We initialize the individuals at generation $t = 0$ with different mutation rates in different runs. The experiments of Section 4.1.2 choose $n = 400$ and $\lambda = 500$, as in [19]. When self-adaptation is used, we initialize all $\chi := 0.1$ and all $\varphi := 0.1$. All strategies are initialized sampling uniformly at random from $\{0,1\}^n$. We repeat each experiment at least 50 times.

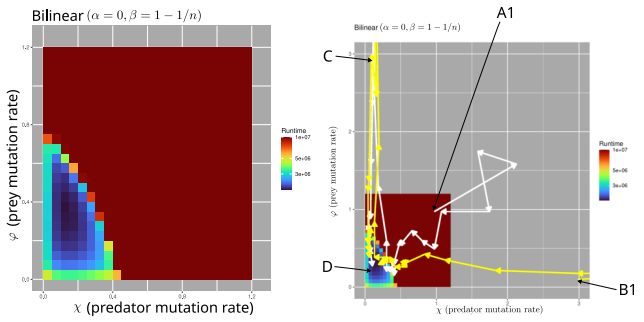
4.1 Results & Discussion

We address RQ-1, as well as parameter sensitivity of the method of mutation rate self-adaptation in Section 4.1.1 by using Bilinear. We address RQ-2 in in Section 4.1.2 and use both Bilinear and DefendIt-B. RQ-3 is addressed in Section 4.1.3 with DefendIt-B.

The experiments of Section 4.1.2 investigate the dynamics of SA-PDCoEA with an emphasis on the potential benefits of using different mutation rates throughout the optimization.

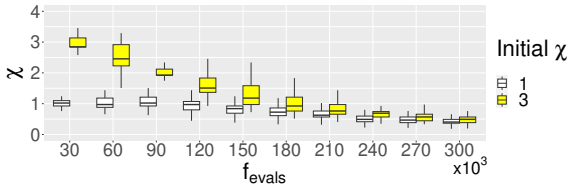
4.1.1 Adaptation Towards Empirical Error Threshold Values (RQ-1). Figure 3a shows how the runtime of the PDCoEA (static mutation rates, S-S) applied to the Bilinear problem ($\alpha = 0, \beta = 1 - 1/n$) depends on the predator and prey mutation rates. The color indicates the runtime (number of fitness evaluations) for static mutation values to reach the optima, with dark blue having the lowest average runtime and dark red having an average runtime greater than 10^7.

In Figure 3b (overlaying Figure 3a) we show mutation rate trajectories of two SA-PDCoEA runs under experimental design A-A (both populations adapt mutation rate) for the same Bilinear problem instance but different initiatlizations of the mutation rate. Each run is of a different color: yellow or white. Arrows are added to assist with tracing. *A1* and *B1*: show mutation rates at the start of each run. *C*: shows mutation rates in the middle of the run. *D*: shows mutation rates at the end of the run. They are lying within the parameter region (dark blue that exhibits low runtime, as observed by the static PDCoEA. We repeated the experiment 50 times for both initial mutation rates to analyze the distribution of predator mutation rates as a function of time. Figure 3c shows the analysis. For both initialization values, the predator mutation rate converges towards values consistent with the optimal values observed for the PDCoEA. This provides empirical evidence that SA-PDCoEA is able to adapt the mutation rate to the values found by the empirical error threshold with PDCoEA. According to the non-parametric test (Wilcoxon rank-sum with Bonferroni correction) applied to



(a) Left: Runtime of PDCoEA with static predator and prey mutation rates on Bilinear ($\alpha = 0$ and $\beta = 1 - 1/n$). Each pixel is the average runtime over 50 runs for a fixed pair of predator (χ on X-axis) and prey mutation rates (φ on Y-axis). Dark blue is lowest average runtime, dark red, highest average runtime $> 10^7$. Mutation rates ranged from 0.1 to 1.2.

(b) Right: Mutation rate dynamics in two runs (yellow run and white run) of SA-PDCoEA applied to Bilinear ($\alpha = 0$ and $\beta = 1 - 1/n$). The yellow run was initiated with mutation rates $\chi = 3$ and $\varphi = 0.1$, and the white run was initiated with $\chi = 1$ and $\varphi = 1$. Lines plot maximum mutation rates in the populations. The X-axis is the predator mutation rate (χ), and the Y-axis is the prey mutation rate (φ). The background shows the runtime of the PDCoEA with static mutation rates (see (a) above). *A1* and *B1*: Mutation rates at the start of the run. *C*: Mutation rates in the middle of the run. *D*: Mutation rates at the end of the run lying within the parameter region giving low runtime of PDCoEA.



(c) Predator mutation rate dynamics of SA-PDCoEA over 100 runs, starting with mutation rates $\chi = \varphi = 1$ and with $\chi = 3, \varphi = 0.1$.

Figure 3: PDCoEA runtime and SA-PDCoEA mutation dynamics.

the predator mutation dynamics, during the first 12000 evaluations, the mutation rate shows no significant variation. After that, the mutation rates significantly change every 6000 evaluation functions. The mutation rate trajectories also show that different mutation rates can be beneficial for different parts of the search space.

Self-Adaptation Method Evaluation. We evaluate the self-adapation method on two factors. First, we can confirm that the method is flexible enough that, for each of the two problem environments, the dynamics of the rate are unique to each one. Second, we investigate how method parameters p_{inc} and A influence the performance of Bilinear, see Figure 4. We observe that performance favors setting the probability of increasing or decreasing the mutation rate to $p_{\text{inc}} \leq 0.5$ when the change factor A is large, i.e., ≥ 2.25. With $p_{\text{inc}} > 0.6$ both $A \leq 1.05$ and $A \geq 7.25$ perform poorly. Performance is also poorer with $p_{\text{inc}} \leq 0.4$ and $A \leq 1.25$. From within the narrow zone of better performance we choose the combination of $p_{\text{inc}} = 0.5$ and $A = 1.05$ for our experiments.

4.1.2 Adaptation Dynamics (RQ-2). We analyze the dynamics of SA-PDCoEA in two dynamic problem environments using four different self-adaptation designs (Table 1).

Bilinear. Figure 5 shows the median worst-payoff in the populations and mutation value dynamics for Bilinear for different experimental designs. Table 2 shows the payoff of the maximin-optima for reference.

We first consider the most complex experiment (A-A) where both populations adapt mutation rates. Figure 5a shows that the

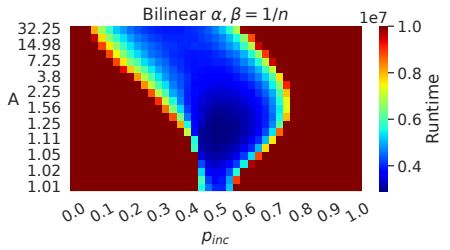


Figure 4: Heatmap of A and p_{inc} parameter sensitivity on the BILINEAR problem. Y-axis is A values in range $[1.01, \ldots, 32.25]$. X-axis is p_{inc} values in range $[0.0, \ldots, 1.0]$. Cell shows the number of fitness evaluations to reach the optima $\alpha = \beta = 1/n$.

Table 2: Payoff for the maximin-optimal solutions on BILINEAR.

BILINEAR parameters	Predator	Prey
$\alpha = 1/n\ \beta = 1/n$	$-1/n^2 \approx 0$	$1/n^2 \approx 0$
$\alpha = 1/2\ \beta = 1/2$	$-1/4$	$1/4$
$\alpha = 1 - 1/n\ \beta = 1 - 1/n$	$-1 - 1/n^2 + 2/n \approx -1$	$1 + 1/n^2 - 2/n \approx 1$

payoff converges to the maximin-optimal values. In addition, at the beginning of the optimization, and twice when the environment's optimum changes (parameters α and β), the mutation rate readjusts. It initially spikes but later settles down towards the minimum value. A possible explanation is that while the algorithm is searching the optimum, a larger mutation rate appropriately favors exploration. Once the maximin-optima has been reached, evolution reduces the mutation rate to maintain the optimal solutions.

We make a comparison to the baseline experimental design, see Figure 5b, where neither population adapts the mutation (Static-Static, S-S). Different combinations of static mutation rates lead to different payoffs and convergence times. We note that, in the first and third phase when mutation rates (e.g., $\chi = 0.8, \varphi = 0.8$) are too high, the result is payoffs that are not the maximin-optimal values. We believe that this is because in these phases the maximin-optimal solutions need to have exactly one 1-bit or 0-bit and higher mutation rates "destroy" good solutions. On the other hand, higher mutation rates converge faster in the second phase than smaller mutation rates. This indicates that there is no unique optimal mutation rate for the whole optimization process. Instead, it is beneficial to use different mutation rates at different times.

When only one population adapts its mutation rate, i.e., experimental designs A-S and S-A, the dynamics shed the complexity of rate co-adaptation. In Figure 5d (A-S) we see more differences in the adaptive mutation rate when the adversary has different static values. We see that low and high static mutation rates for the prey instigate small adaptive mutation rates for the predator and medium static mutation rates for the prey instigate high adaptive mutation rates for the predator. In contrast this is not observed in Figure 5c (S-A). We conjecture that this is because of a unique dynamic when optimizing BILINEAR with $\alpha = \beta = 1/n$ and $\alpha = \beta = 1 - 1/n$. We will explain the hypothesized dynamic for $\alpha = \beta = 1/n$ but we believe that there is a similar dynamic for $\alpha = \beta = 1 - 1/n$.

If α=β=$1/n$ and there is a sufficiently large proportion of solutions in the prey population with more than one 1-bit, it is profitable for the predator population to increase the number of 1-bits and the opposite is true when there is a large proportion of solutions with less than one 1-bit. In turn solutions with more than one 1-bit in the predator population increase the evolutionary pressure for the prey population to reduce the number of 1-bits and vice versa. Then, our hypothesis is that for high static mutation rates in Figure 5d (A-S) the prey population cannot create solutions with exactly or less than one 1-bit and the predators always want to increase the number of 1-bits; once the all-ones bitstring is reached the mutation rate decreases to keep this solution. A similar thing happens for high static mutation rates in Figure 5c (S-A).

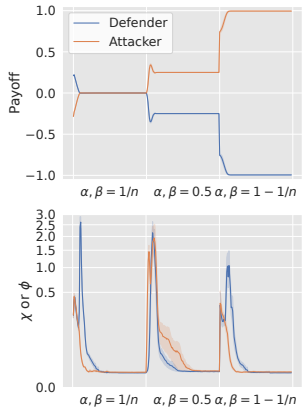
The hypothesis for medium mutation rates in Figure 5d (A-S) is that the prey population can create solutions with less or exactly one 1-bit, but also some solutions with more than one 1-bit. Most of the time this proportion is small so the predator population benefits from reducing the number of 1-bits and most of the population have small number of 1-bits and low mutation rates to keep these solutions. However, this proportion of solution in the prey population is sometimes large enough for the predator population to tend to increase the number of 1-bits; since there are many 0-bits high mutation rates are beneficial. Then, the predator population switches between increasing and decreasing the number of 1-bits which in turn increases and decreases the mutation rates sharply. On the other hand, for medium mutation rates in Figure 5c (S-A) when the predators (with static mutation) create a large proportion of solutions with more than one 1-bit the prey are pushed to reduce the number of 1-bits but since they already have few 1-bits the mutation rates stay small. Finally, for low static mutation rates in the runs of Figures 5d (A-S) and 5c (S-A), the algorithm may be able to maintain most solutions with exactly one 1-bit for both predator and prey populations avoiding this dynamic.

DefendIt-B. Figure 6 shows the median worst-payoff and mutation value dynamics for DefendIt-B for each experimental design. For adaptive adversaries (A-A), Figure 6a, the payoff decreases for the defender as the budget increases. This could be explained by the attacker having access to more actions and using them to acquire more resources. The payoff seems to converge for B_{low} and B_{high}. The larger payoff fluctuations in B_{med} could come from the impact of infeasible solutions. The mutation rate values converge for B_{low} and fluctuate for B_{med} and B_{high}. The mutation rate dynamics for attacker and defender look quite similar.

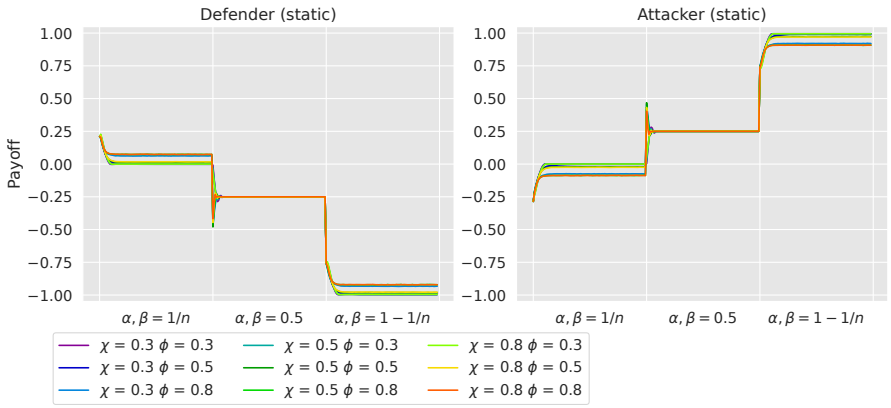
In the non-adaptive baseline, S-S, see Figure 6b, we see that different combinations of mutation rates lead to different payoffs, e.g., mutation rates ($\chi = 0.8, \varphi = 0.8$) that are too high lead to lower payoff when the budget is B_{low} or B_{med}. When the budget is B_{low} or B_{med} several mutation rate combinations result in the median solutions in the population going over-budget. This highlights that the selection of mutation rates is challenging, while the self-adaptation of SA-PDCoEA can find reasonable solutions and mutation rates.

When one adversary's mutation rate is static, we observe that there are differences in the adaptive mutation rate and payoff. The differences for different static values are large (S-S). Figure 6d (A-S) shows high variance for the static attacker payoff when compared to the adaptive defender, specially for $\varphi \in \{0.3, 0.4, 0.5\}$. This difference could come from these mutation rates having difficulties in maintaining feasible solutions. In addition, we note that the adaptive mutation rate not only adapt to the current budget but also adapt to their opponent's mutation rates, i.e., *co-adaptation* of mutation rates. The S-A case in Figure 6c behaves similarly.

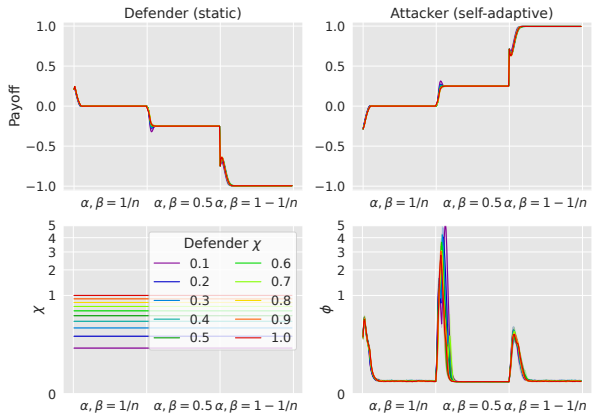
We observe that the mutation rate depends on the budget due to the DefendIt-B problem. With B_{low} there are fewer feasible solutions, so too much change may render a strategy infeasible. With



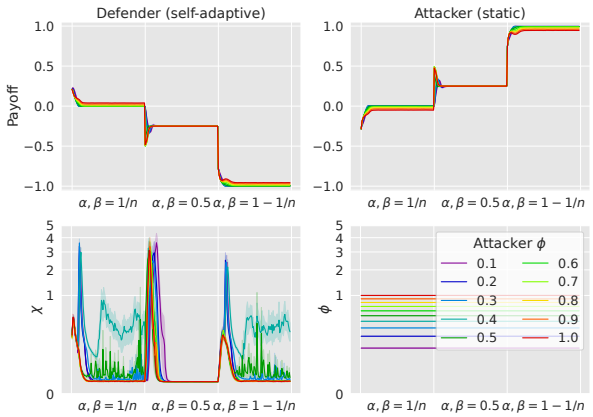
(a) A-A (Line is Attacker/Defender)



(b) S-S (Column is Defender and Attacker. Line is mutation rate combinations for attacker and defender)



(c) S-A (Column is Defender and Attacker. Line is mutation rates for defender)



(d) A-S (Column is Defender and Attacker. Line is mutation rates for attacker)

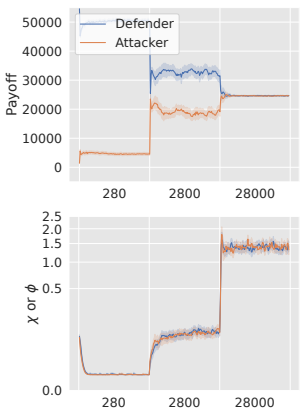
Figure 5: The BILINEAR problem for different mutation rate adaptation combinations: (a) A-A, (b) S-S, (c) S-A, and (d) A-S. X-axis shows the number of fitness evaluations (t) and the location of the optima. Y-axis for top row shows the median maximin payoff. Y-axis for bottom shows χ/φ, note the scale.

B_{med} there are more feasible solutions, and changing a strategy can thus provide more advantage. For B_{high} there are even more feasible strategies, and changing is a even more advantageous.

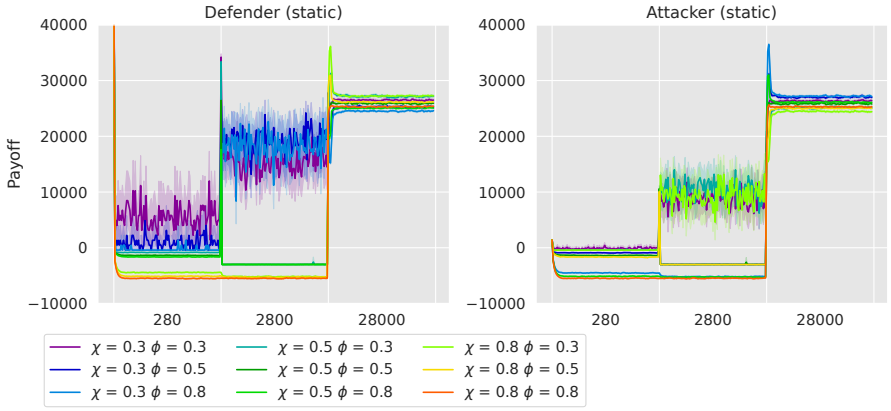
4.1.3 Adaptation Performance (RQ-3). This section analyzes the performance of SA-PDCoEA in each problem environments and experimental design, see Figure 7. Static predator and adaptive prey mutation rate (S-A) at the end of each budget regime in DefendIt-B (Low budget Figure 7a, medium budget Figure 7b, and high budget Figure 7c). Note that the full trajectory is shown in Figure 6d and these are snapshot at different fitness evaluations $t = 1.98e7$, $t = 3.98e7$ and $t = 5.98e7$. The 3rd quantile payoff values for static and adaptive runs is shown as well as the φ (prey mutation rate). We observe that the adaptive mutation values roughly correspond to the good values seen from the static (S-S) settings. The self-adaptive payoff and mutation rates avoid poor payoff and mutation rates when compared to the static values. We observe significant differences in payoff for some mutation values, see A. The BILINEAR results (omitted by page limit) confirm the problem dependency of the mutation rate and further demonstrate the ability of SA-PDCoEA to adapt to an empirically good mutation rate.

5 CONCLUSION

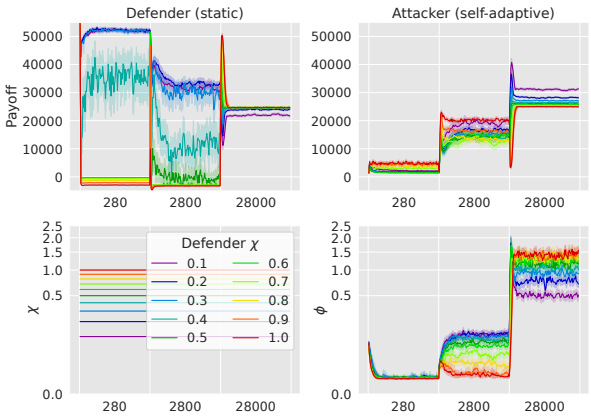
We investigated the impact of self-adapting mutation rates for CCAs by introducing a new algorithm, SA-PDCoEA. One motivation is that prior work has shown the mutation rates to be important to algorithm performance and self-adapting rates avoid the challenge of selecting ideal ones. An intuition is that the inherent non-stationarity of an evolving adversarial population bears some similarity to the dynamic environments in classical EAs that are well served by self-adaption. We observed the performance and dynamics of SA-PDCoEA in in two problem environments that we changed to be dynamic.The broad take-away is that self-adapation avoids bad results that would follow from poor mutation rate choices. It allows the mutation rate to hone in on an optimal setting and adjust to environmental change. The price for this flexibilty and "customize-ability" is lower payoffs leading up to finding a good rate. If an ideal rate was feasible and the algorithm run with it, payoffs would usually be higher. The cost of flexibility is the fitness evaluations and low performance incurred in the course of adaptation. Stepping out, beyond the technical details, self-adaptation introduces more complexity to an already very complex algorithm making analysis very challenging. Future work will investigate additional parameter settings and problems. We will also investigate



(a) A-A (Line is Attacker/Defender)



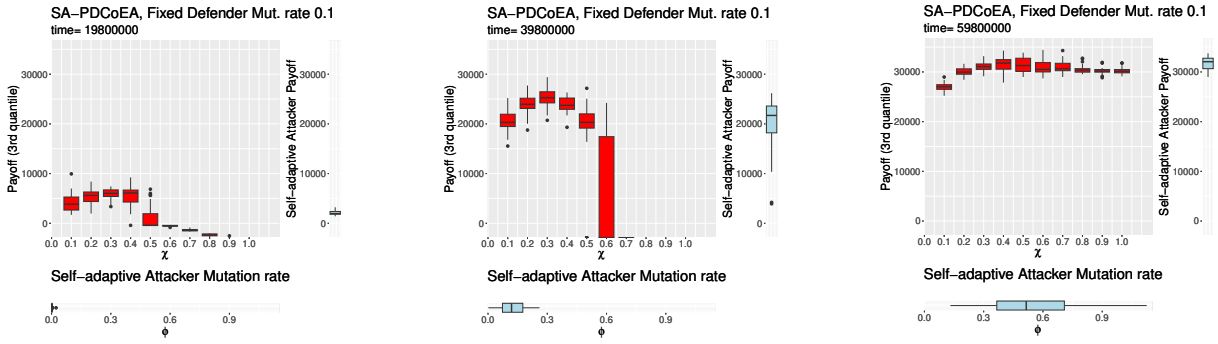
(b) S-S (Column is Defender and Attacker. Line is mutation rate combinations for attacker and defender)



(c) S-A (Column is Defender and Attacker. Line is mutation rates)

(d) A-S (Column is Defender and Attacker. Line is mutation rates)

Figure 6: The DefendIt-B problem for different mutation rate adaptation combinations: (a) A-A, (b) S-S, (c) S-A, and (d) A-S. X-axis shows the number of fitness evaluations (t) and the allowed budget C. Y-axis for top row shows the median maximin payoff. Y-axis for bottom shows χ/φ, note the scale. Left column is defender and right is attacker.



(a) $t = 1.98e7$, B_{low} (b) $t = 3.98e7$, B_{med} (c) $t = 5.98e7$, B_{high}

Figure 7: Static predator and adaptive prey mutation rate (S-A) at the end of each budget regime in DefendIt-B and $\chi = 0.1$ (predator mutation rate). Y-axis is the 3rd quantile payoff, leftmost is the static and right most is the adaptive. X-axis is the φ (prey mutation rate), top is for static φ values and bottom is a boxplot of the adaptive φ values.

how well the solutions generalize to unseen adversaries, not only compare the subjective payoff values.

ACKNOWLEDGMENTS

Lehre and Hevia Fajardo were supported by a Turing AI Fellowship (EPSRC grant ref EP/V025562/1). The computations were performed using the University of Birmingham's BlueBEAR HPC service. See http://www.birmingham.ac.uk/bear for more details. Toutouh was supported by the University of Malaga and TAILOR ICT-48 Network (No 952215) funded by EU Horizon 2020 research and innovation programme. Hemberg and O'Reilly acknowledge funding for this work under US Government Contract #FA8075-18-D-0008.

REFERENCES

[1] Peter J. Angeline and Jordan B. Pollack. 1993. Competitive environments evolve better solutions for complex tasks. In *Proceedings of the Fifth International Conference (GA93), Genetic Algorithms*. 264–270.

[2] L. M. Antonio and C. A. C. Coello. 2018. Coevolutionary Multi-objective Evolutionary Algorithms: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation* (2018), 1–16. https://doi.org/10.1109/TEVC.2017.2767023

[3] Robert Axelrod. 1984. *The Evolution of Cooperation*. Basic, NY, New York.

[4] Thomas Bäck. 1992. Self–Adaptation in Genetic Algorithms. In *Proceedings of the First European Conference on Artificial Life*.

[5] A. B. Cardona, J. Togelius, and M. J. Nelson. 2013. Competitive coevolution in Ms. Pac-Man. In *2013 IEEE Congress on Evolutionary Computation*. 1403–1410.

[6] Brendan Case and Per Kristian Lehre. 2020. Self-Adaptation in Nonelitist Evolutionary Algorithms on Discrete Problems With Unknown Structure. *IEEE Transactions on Evolutionary Computation* 24, 4 (2020), 650–663. https://doi.org/10.1109/TEVC.2020.2985450

[7] Duc-Cuong Dang and Per Kristian Lehre. 2016. Self-adaptation of Mutation Rates in Non-elitist Populations. In *Parallel Problem Solving from Nature – PPSN XIV (Lecture Notes in Computer Science)*. Springer, Cham, 803–813. https://doi.org/10.1007/978-3-319-45823-6_75

[8] Benjamin Doerr, Carsten Witt, and Jing Yang. 2018. Runtime analysis for self-adaptive mutation rates. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Kyoto, Japan) *(GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 1475–1482.

[9] Paul R Ehrlich and Peter H Raven. 1964. Butterflies and plants: a study in coevolution. *Evolution* 18, 4 (1964), 586–608.

[10] Sevan Gregory Ficici. 2004. *Solution concepts in coevolutionary algorithms*. Ph. D. Dissertation. Brandeis University.

[11] D Fogel. 2001. Blondie24: Playing at the Edge of Artificial Intelligence.

[12] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[13] Erik Hemberg, Jacob Rosen, Geoff Warner, Sanith Wijesinghe, and Una-May O'Reilly. 2016. Detecting tax evasion: a co-evolutionary approach. *Artificial Intelligence and Law* 24 (2016), 149–182.

[14] Mario Hevia Fajardo, Per Kristian Lehre, Erik Hemberg, Jamal Toutouh, and Una-May O'Reilly. 2023. Analysis of a Pairwise Dominance Coevolutionary Algorithm with Spatial Topology. In *Genetic Programming Theory and Practice XXX*. Springer.

[15] Stephen T Jones, Alexander V Outkin, Jared Lee Gearhart, Jacob Aaron Hobbs, John Daniel Siirola, Cynthia A Phillips, Stephen Joseph Verzi, Daniel Tauritz, Samuel A Mulder, and Asmeret Bier Naugle. 2015. *Evaluating moving target defense with pladd*. Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[16] Krzysztof Krawiec and Malcolm Heywood. 2016. Solving Complex Problems with Coevolutionary Algorithms. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. ACM, 687–713.

[17] Per Kristian Lehre. 2010. Negative Drift in Populations. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN 2010) (LNCS, Vol. 6238)*. Springer Berlin / Heidelberg, 244–253. https://doi.org/10.1007/978-3-642-15844-5_25

[18] Per Kristian Lehre. 2022. Runtime Analysis of Competitive Co-Evolutionary Algorithms for Maximin Optimisation of a Bilinear Function. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 1408–1416. https://doi.org/10.1145/3512290.3528853

[19] Per Kristian Lehre, Mario Hevia Fajardo, Erik Hemberg, Jamal Toutouh, and Una-May O'Reilly. 2023. Analysis of a Pairwise Dominance Coevolutionary Algorithm And DefendIt. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '23)*. Association for Computing Machinery, New York, NY, USA, 9 pages.

[20] Chong-U Lim, Robin Baumgarten, and Simon Colton. 2010. Evolving behaviour trees for the commercial game DEFCON. In *European Conference on the Applications of Evolutionary Computation*. Springer, 100–110.

[21] Sean Luke et al. 1998. Genetic programming produced competitive soccer softbot teams for robocup97. *Genetic Programming* 1998 (1998), 214–222.

[22] Silja Meyer-Nieberg and Hans-Georg Beyer. 2007. Self-Adaptation in Evolutionary Algorithms. In *Parameter Setting in Evolutionary Algorithms*, Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz (Eds.). Vol. 54. Springer Berlin Heidelberg, Berlin, Heidelberg, 47–75. https://doi.org/10.1007/978-3-540-69432-8_3 Series Title: Studies in Computational Intelligence.

[23] Melanie Mitchell. 2006. Coevolutionary learning with spatially distributed populations. *Computational intelligence: principles and practice* 400 (2006).

[24] Gabriela Ochoa. 2006. Error Thresholds in Genetic Algorithms. *Evolutionary Computation* 14, 2 (June 2006), 157–182. https://doi.org/10.1162/evco.2006.14.2.157

[25] Una-May O'Reilly, Jamal Toutouh, Marcos Pertierra, Daniel Prado Sanchez, Dennis Garcia, Anthony Erb Luogo, Jonathan Kelly, and Erik Hemberg. 2020. Adversarial genetic programming for cyber security: A rising application domain where GP matters. *Genetic Programming and Evolvable Machines* 21 (2020), 219–250.

[26] Elena Popovici, Anthony Bucci, R. Paul Wiegand, and Edwin D. De Jong. 2012. *Coevolutionary Principles*. Springer Berlin Heidelberg, Berlin, Heidelberg, 987–1033.

[27] Christopher D Rosin and Richard K Belew. 1997. New methods for competitive coevolution. *Evolutionary Computation* 5, 1 (1997), 1–29.

[28] Hans-Paul Schwefel. 1974. *Adaptive Mechanismen in der biologischen Evolution und ihr Einfluß auf die Evolutionsgeschwindgkeit*. Technical Report. Technical University of Berlin.

[29] Karl Sims. 1994. Evolving 3D morphology and behavior by competition. *Artificial life* 1, 4 (1994), 353–372.

[30] J. Togelius, P. Burrow, and S. M. Lucas. 2007. Multi-population competitive co-evolution of car racing controllers. In *2007 IEEE Congress on Evolutionary Computation*. 4043–4050.